Blackjack as a Testbed for Reinforcement Learning: Monte Carlo, Q-Learning, and Double Q-Learning Under Realistic Conditions

Rohan Sanjay Patil
Master Artificial Intelligence, Faculty of Computer Science
Technical University of Applied Sciences, Würzburg-Schweinfurt
rohansanjay.patil@study.thws.de

***Abstract*—Blackjack. A game played by millions but won by only a few. What if a computer played Blackjack for us? Can it learn the best strategy to win the game? How good would it be against the dealer? Let us find out. In this paper the goal is to describe the implementation of a real-world BlackJack by teaching it the rules right from the scratch. Using multiple methods like Monte Carlo, Q-Learning and a variation of the same called Double Q-Learning. And how to check if the algorithm's performance is up to the mark? By testing it with basic strategy conditions, Hi-Lo card counting system and multiple rule variation scenarios. And then comparing it with the general idea of how the algorithm is supposed work in the real world. The experimental results depict how Double Q-learning consistently outperforms other methods. The task is also to try and find out the difficulties in achieving stable Q-learning values due to complex play through of Blackjack, and multiple hyperparameters in the system. The key findings also indicate how card counting strategy implemented provided marginal improvement over basic strategy but added a very complex nature, which is not worth all the computation. The addition of rule variations in this system clearly depicts how a single change in the game rules could tip the scale of advantage from dealer to the player and vice-versa.**

***Index Terms*—Reinforcement Learning, Q-Learning, Monte Carlo Methods, Blackjack Strategy, Hi-Lo Card Counting, Policy Optimization, Markov Decision Processes**

I. Introduction

Blackjack, an interesting blend of mathematical probability and chance, and this has sparked interest in both casual players and professional gamblers for ages. What if, instead of relying on human intuition or memorized tables, an AI agent was assigned to discover the best way to play from scratch by learning only by experience? This was the motivating question for the research, where the world of reinforcement learning (RL) [1] and casino strategy were brought together to see if an algorithm can truly "learn to win."

The core of this research is a highly configurable, Python-based RL simulation of Blackjack. The designed environment is here to replicate the real-world casino play as closely as possible. Notably, the simulator introduces:

- Multiple decks and realistic card penetration (reshuffling),
- Rule toggles for crucial options like late surrender, doubling, and "player blackjack always wins",
- Card counting supports a card counting system called the Hi-Lo system [2], delivered directly as part of the agent's state,
- Five possible player actions including Surrender and Double Down, with context-aware masking.

This rich environment is not only more faithful to casinos than most academic examples, but also serves as a flexible testbed for RL research and experimentation.

This paper sets out to answer several fundamental questions:

- Can classic RL algorithms discover optimal or near-optimal Blackjack play from experience alone?
- How does choosing between Monte Carlo, Q Learning and Double Q learning affect the efficiency and also the robustness of the environment?
- Does providing a counting variable help with aggressive betting and counting like us humans [2]?
- How do the critical rule changes (e.g., adding an option to surrender) reshape learned policies? Do agents also adapt like us [3]?

The methodology puts three RL algorithms—Monte Carlo Control [4], Q-Learning [5], and Double Q-Learning [6]—through their paces. Training occurs over hundreds of thousands to millions of hands, in four distinct environments:

- Basic Strategy (vanilla Blackjack) [7],
- Card Counting (Hi-Lo system, with variable betting) [2],
- Late Surrender enabled [3], and
- Player-21-always-wins.

By analyzing game outcomes, policy heatmaps, learning curves, and agent decision patterns, it was concluded that Double Q-Learning consistently produces the most stable and human-like strategies, besting standard Q-Learning and Monte Carlo Control. Moreover, the practical benefit of card counting in an RL setup was found to be limited: the agent's learning cost (in terms of state-action pairs and training time) often outweighed real-world gains [2], mirroring the intense complexity of card counting in casinos.

This research not only demonstrates the power and limits of RL in Blackjack but also highlights how the algorithmic bias, state space design, and the gritty details of real casino rules come into play. Along the way, everyone can learn how even Blackjack can challenge the design of AI based learning systems [8], [9].

II. Description of Reinforcement Learning Player

A. Theoretical Background

The paper decided to Model the implementation of Blackjack based on the well known Markov Decision Process (MDP) [1] because at it's core, Blackjack is a sequential decision making problem: the player receives some information (the cards in their hands, initially 2, the dealer's up-card, and a card count signal in their head) and the player must select one action from a set of various actions to maximize his reward and gain profit, and that is a natural fit for the Markov Decision Process framework.

States in this MDP encapsulate all relevant, observable information at each decision-point. In this environment, a state is defined as:

- The player's hand value (ranging from 4 to 21),
- The dealer's visible up-card (2 to 11),
- A flag for a usable ace (soft hand indicator, 0 or 1),
- And—if card counting is used—the binned Hi-Lo "true count" signal [2] (from -2 to +2).

Actions modeled are those available to a player at a real table:

Stand, Hit, Double Down, Split, and Surrender [7] (governed by enabled rules).

Transitions are fully governed by Blackjack mechanics, combining deterministic rules (dealer must hit until 17, etc.) and stochasticity from the shuffled multi-deck shoe, which the environment models with re-shuffling determined by a penetration setting.

Rewards are granted solely at a terminal state—after the hand is decided. This makes Blackjack an episodic task with sparse rewards, where typical RL approaches that rely on frequent feedback are severely challenged [8].

$$\mathbb{E}[\text{Reward}] = b \cdot (p_{\text{win}} \cdot r_{\text{win}} + p_{\text{lose}} \cdot r_{\text{lose}} + p_{\text{push}} \cdot r_{\text{push}}) \quad (1)$$

In Blackjack:
$r_{\text{win}} = 1$ (or 1.5 for natural)
$r_{\text{lose}} = -1$
$r_{\text{push}} = 0$

Evaluated RL Algorithms: For evaluation an empirical comparison between three classic, tabular reinforcement learning algorithms—each representing a distinct approach to solving MDPs:

- **Monte Carlo Control**: This method learns Q-values based on full-episode returns, updating the estimated action-values only once a hand is complete. It's sample-inefficient but unbiased [4], and especially suitable for episodic games. Here, an ϵ-greedy policy (ϵ = 0.1) ensures continued exploration. For each first-visit (s_t, a_t) within an episode, the update is:

$$Q(s_t, a_t) \leftarrow [Q(s_t, a_t) + \alpha\big[G_t - Q(s_t, a_t)\big]] \quad (2)$$

where the return is

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \quad (3)$$

α: learning rate (a fixed constant)
γ: discount factor (usually 1.0 for episodic Blackjack)
G_t: observed sum of discounted rewards from step t You only update Q(s,a) for the first occurrence of (s,a) in each episode.

- **Q-Learning**: This temporal-difference (TD) algorithm instead updates its Q-values at every step, bootstrapping from future estimated values. With a learning rate (α) of 0.05 and discount γ = 0.99, and a decaying γ (down to 0.05), Q-Learning seeks to quickly propagate information about high-reward terminal states back to earlier actions [5]. It's a proven baseline for discrete-action RL tasks. After every transition $(s_t, a_t, r_{t+1}, s_{t+1})$, update:

$$Q(s_t, a_t) \leftarrow [Q(s_t, a_t) + \alpha\big[\, r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\,\big]] \quad (4)$$

s_t: current state
a_t: action taken
$rt + 1$: observed reward after action
$st + 1$: next state
γ: discount factor
α: learning rate

- **Double Q-Learning**: This improvement uses two separate Q-tables to decouple action selection from evaluation, mitigating Q-Learning's classic overestimation bias (especially problematic in noisy, sparse environments like Blackjack) [6]. Implementing an aggressive exploration strategy (lower α=0.005) while retaining γ=0.99. Maintain two value functions Q^A and Q^B. At each step, with probability 0.5 update one randomly
This progression—from episodic-search to bootstrapping, to bias-correction—forms a natural ladder across RL sophistication and stability.

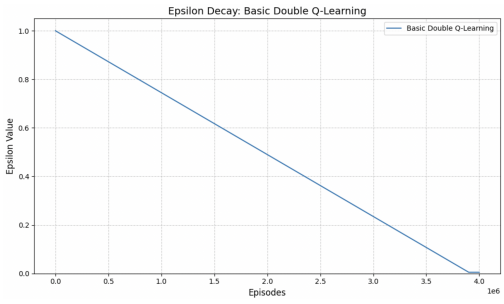

Fig. 1. Epsilon Decay for Double-Q Learning vs Episodes

B. State and Action Space

The program's state space is carefully constructed to balance informativeness and tractability:

- Player hand value: 4 to 21 (since anything below 4 is impossible because there are 2 cards, and value above 21 means the player/dealer lost and that is a terminal state),
- Dealer up-card: 2–11 (with 11 representing Ace),

– Usable ace indicator (does the player have a "soft" total? This can dramatically change optimal action!),
– For card-counting environments: a binned "true count" [2], calculated by normalizing the running Hi-Lo count by decks remaining and discretizing into -2, -1, 0, +1, +2 for practical state representation.

Combinatorially, this creates:

– For basic: ∼1,800 states (4–21 player hand × 2–11 upcard × 0/1 usable ace)
– For counting: ∼9,000 states (adds 5 count bins)

$$N_{\text{states}} = N_{\text{player}} \times N_{\text{dealer}} \times N_{\text{usableAce}} \times N_{\text{countBins}} \quad (5)$$

Each state supports a discrete action space of up to 5 actions (subject to rule constraints and valid-move masking).

The actions:

– Stand: End the turn, dealer resolves.
– Hit: Draw another card.
– Double : Double the initial bet, draw only one card (if allowed).
– Split: For equal cards (more complex, typically omitted/approximated in the code).
– Surrender: Forfeit the player's hand before the actual end of the round (taking only half the loss), if rule is active and timing is valid [7].

The well-designed state and action space provides the RL agent the flexibility to discover realistic human-like or even superhuman strategies, but also explodes the space that must be explored, that is the prime challenge.

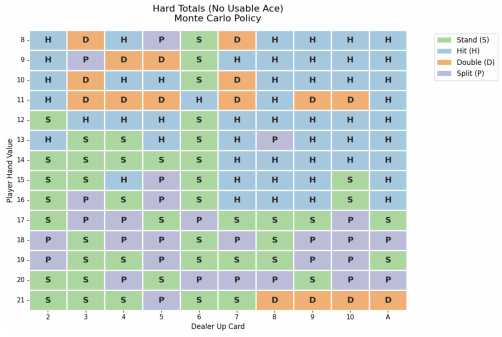

Fig. 2. Basic strategy Heatmap by MC(Hard Totals)

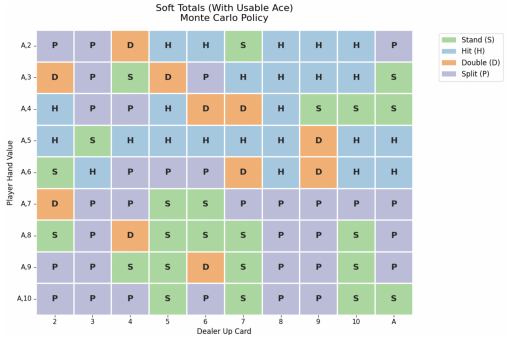

Fig. 3. Basic strategy Heatmap by MC(Soft Totals)

C. Environment Configurations

The notebook and codebase enable us to "flip switches" which replicate various real casino rules:

– **Basic Strategy Environment**: Classic blackjack, core rules only, no frills [7]. This is the essential baseline and tests if RL can match established strategy tables.
– **Card Counting (Hi-Lo) Environment**: There is an important additional state in the system, that is, the binned true count [2] and these bets can scale by count, mimicking advantage player techniques. The RL agent can theoretically discover that "hot" counts (lots of tens/aces left) merit higher bets. This is a classic intersection of RL and human gambling strategy research. The bet size b is determined by discretized true count bins, such that:

$$b = \begin{cases} 1, & \text{if } C \leq 0 \\ 4, & \text{if } 1 \leq C < 2 \\ 8, & \text{if } C \geq 2 \end{cases} \quad (6)$$

where C is the binned true count:

$$C = \text{bin}\left(\frac{\text{running count}}{\text{decks remaining}}\right) \quad (7)$$

and $\text{bin}(\cdot)$ maps the true count to one of the discrete bins used in the agent's state representation.
– **Late Surrender Environment**: Surrender becomes an additional action to the space(with a condition that it can be used only if you have not used a Hit in the round, and only if the dealer does not have a natural blackjack, the code checks for this) [3]. If used optimally, this rule can reduce loss rates in adverse situations (e.g., hard 16 vs. dealer 10). Evaluating if and when RL agents learn this is highly informative.
– **Player 21 Always Wins**: In some casino variants, player blackjack always beats the dealer—even if the dealer has 21. This rule is rare but tilts the game slightly toward the player [7]. The environment implements this flexibly, providing insight into how such tweaks alter learning and final policies.

By modeling blackjack with such attention to state richness, action realism, rule flexibility, and the inclusion of explicit card counting, this project provides a true

"sandbox" for reinforcement learning research. Learning in this environment isn't just a test of algorithmic prowess; it's a study of what it takes for computers to outwit casinos on their own terms. The theoretical structure set here underpins everything in the experiments and results.

III. Experiments and Evaluation

A. Experimental Setup

To exclusively evaluate the agent's performance, an artificial pipeline was designed that replicates professional RL evaluation procedure [10] and common casino math logic:

Training model:: Each RL algorithm that was implemented, namely Monte Carlo Control, Q-Learning, and Double Q-Learning were all trained from scratch in a specific variant of the environment. The total training duration/training episodes are assigned from only 100,000 episodes (Monte Carlo with basic rules) up to 5,000,000 episodes ((double)Q-Learning agents with card counting environment). The episode count was scaled according to environment complexity and learning stability as that is a necessity, where non-trivial increases for counting agent lead them to actual convergence in the state and action space.

Evaluation Protocol:: After training, each agent was tested in a frozen environment using a fixed policy. Evaluation was always performed over 500,000 independent Blackjack hands using the learned policy in each environment, to ensure statistical reliability and minimize variance.

Metrics Tracked::

- Average reward per hand,
- Win, loss, and push rates,
- Epsilon Decay per Episode,
- Surrender choice rates (for relevant rules).

Environments Modeled::

- Basic strategy (Normal Blackjack with no extra rules/variation)
- Card counting (Hi-Lo system and variable bets)
- Late surrender (An option to surrender in the middle of the game)
- Player 21 always wins (player always wins on 21, might lead to more average reward for the player)

All experiments were done with a 6 deck shoe, penetration of 50% (cards reshuffled after half of the shoe is used up). Key learning hyperparameters for every algorithm were chosen via monitoring learning curves (not grid search), with preference for policy stability and convergence.

Stochasticity Handling and Reproducibility:

- Fixed random seeds in experiments at training and test time to make results reproducible.
- There was no experience replay or batch update used; all learning was per-episode or on-policy.
- Illegal moves during training (e.g., surrender when not permitted) were internally mapped to the closest legal action (usually "stand").

B. Results and Discussion

1) Algorithm Comparison: The following main findings were seen in the vanilla (standard rules) game and while card counting:

Interpretation:

- Double Q-Learning consistently attained both the highest average reward and the most stable policy, particularly as state-space complexity increased (e.g., under card counting) [6].
- Q-Learning performed much better than Monte Carlo in simple environments, yet not all learning carried over to the card counting situation—demonstrating the method's infamous propensity for overestimating values or breaking down under sparse-reward noise [5], [8].
- Monte Carlo Control—while easy to implement—lagged behind due to its slow, variance-heavy credit assignment; nonetheless, it responded better to variable rewards under card counting, perhaps due to unbiased value estimation [4].

Model	Strategy	Win Rate	Avg. Reward
Monte Carlo	Basic Strategy	42.29%	-0.0459
Q-Learning	Basic Strategy	42.36%	-0.0447
D-Q-Learning	Basic Strategy	42.75%	-0.0256
Monte Carlo	Card Counting	42.91%	-0.0324
Q-Learning	Card Counting	42.79%	-0.0440
D-Q-Learning	Card Counting	42.97%	-0.0304

TABLE I
Overall Win Rate and Average Reward Comparison Across Strategies

2) Q-Value Stability: **Basic Environments:** With only player total, usable ace, and dealer upcard ($\sim$1,800 states, each with 5 actions; $\sim$9,000 pairs, only some reachable), both Q-learning and Double Q-learning consistently converged to stable policies within 1–2 million rounds. Monte Carlo needed much less, but the learned policy remained noisy and suboptimal.

Card Counting Environments: Including the true count bin increased total states by a factor of five ($\sim$9,000 states, $\sim$45,000 pairs). This caused slower convergence, and made adequate exploration (by epsilon scheduling) essential. Double Q-Learning's decoupling of estimation from policy update clearly lessened the instability Q-Learning suffered—likely because rewards when stakes were high (big bet, rare positive count) were so infrequent that traditional Q-values became unreliable [6], [8].

3) Rule Variations Impact:

Late Surrender Enabled: The agent learned to utilize surrender in about 10.68% of eligible cases. Average reward decreased slightly to -0.0524 (from -0.0459 basic), but analysis of win/loss distribution confirmed loss minimization: rather than losing full bets in hopeless spots, the agent(player) has an option to cut its loss in half [3], [7].

Player 21 Always Wins: Rule change produced a marginal improvement in average reward (-0.0475), associated with modestly more aggressive play when holding soft hands or high ace potential [7].

Ruless	Model	Win %	Avg. Reward	Surrender %
Late Surrender	MC	38.85%	-0.0524	10.64%
Player 21 Wins	MC	42.49%	-0.0475	0.00%

TABLE II
IMPACT OF RULE VARIATIONS ON MONTE CARLO PERFORMANCE

Behavioral Analysis: Action frequency plots and state-action "heatmaps" confirmed:

- Surrender was almost always used only when mathematically justified [3].
- Doubling and splitting clustered in advantageous cases, mirroring human strategy [7].

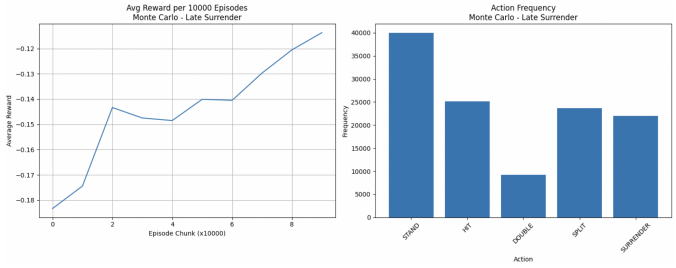

Fig. 4. Late surrender Variation Performance

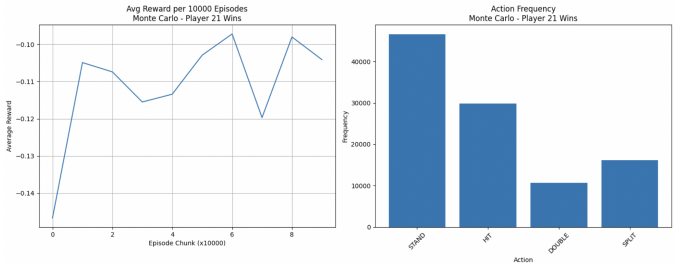

Fig. 5. Player 21 wins Variation Performance

Interpretive Insights:

- *Why does Double Q-Learning excel?* By decoupling greedy selection from value update, DQL avoids chasing "phantom" values caused by rare but large rewards (e.g., max bet with high count) [6]. This makes it robust to variable reward structures.
- *Was card counting worth it?* Computationally, the answer is mixed: while agents did learn to correlate large bets with positive counts [2], the expected gain after immense training was surprisingly slight—mirroring real-world experience, where skillful counting edges can be wiped by house rules, shuffling depth, and variance.
- *Did agents learn human-like policies?* Yes—even without prior knowledge, the best learned behaviors closely tracked published basic strategy charts [7], surrendered only when it was allowed and when there was a high chance of losing, and bet huge amounts when the deck was "hot."

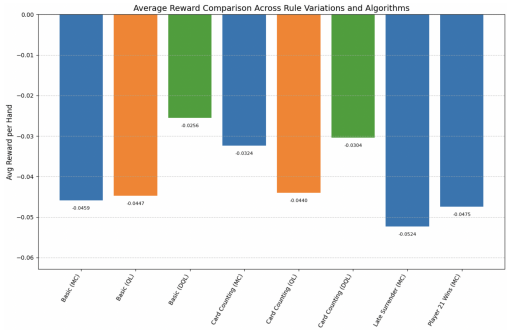

Fig. 6. Average Reward Comparison Across Algorithms

IV. CONCLUSION

This project set out to answer a seemingly simple yet fundamentally deep question: can a reinforcement learning agent, starting from scratch and playing only against chance, discover not just the surface rules but the underlying strategies that make successful Blackjack play? By building a flexible simulation platform, modeling the environment as a robust Markov Decision Process [1], and rigorously evaluating multiple RL algorithms, the research provides several important insights into both RL and practical card game strategy.

First and foremost, Double Q-Learning [6] clearly stands out as the most robust approach across all tested configurations. It consistently achieved the highest average rewards, most stable learning, and most human-reasoned strategies, particularly as the task complexity increased with card counting or new rule introductions. This superiority can be traced to Double Q-Learning's ability to reduce the value overestimation traps that affect classic Q-Learning [5], especially in environments with sparse rewards or high variance, as is typical in Blackjack.

All three algorithms—Monte Carlo [4], Q-Learning [5], Double Q-Learning [6]—proved capable of learning baseline strategy, often matching or approaching the choices made in published "basic strategy" charts [7]. Even without explicit knowledge, agents learned to stand, hit, double, and (where enabled) surrender in the mathematically optimal scenarios. This underscores the expressive power of model-free RL when sufficient exploration is present and the environment is carefully constructed.

Card counting, implemented via the Hi-Lo point-count system [2] and delivered as a discretized state feature, offered interesting (and perhaps unexpectedly realistic) lessons. Although the RL agent learned to raise bets when the deck was "hot," the actual improvement in average return was modest, mirroring the real-life that, while

card counting edges exist, they are both fragile and hard-won. Moreover, the additional space complexity and time required for implementing count was made worth it, but only by Double Q-Learning. It was able to consistently leverage this information for profit.

Rule changes had predictable and instructive effects. When late surrender was enabled [3], the agent used it only in scenarios well-known to minimize losses (such as 'hard 16 vs. dealer 10'), confirming that RL agents not only optimize for expected reward but also adapt their risk profile as rules evolve. The "player 21 always wins" rule gave a small but a slight reward boost [7], which is actually minimal but shows how agent behavior changes even for rare edge-case rules.

Limitations and Future Work: Several simplifications were necessary for this research: the environment models a one player game excluding the dealer, only 1 player against a deterministic dealer, splitting logic is minimal, insurance and real multi-hand play are omitted, and learning is tabular (not neural). As a result, scalability to richer environments, multi-agent games, or real casino nuances is limited. Addressing these simplifications—including multi-agent competitive/cooperative scenarios, deep RL with function approximation [9], and more advanced counting/switching strategies—would be highly valuable. Additionally, practical evaluation in non-stationary, adversarial, or human-opposed environments could deepen the understanding of RL's role in real-world games of chance [4], [8].

In summary: This research shows that reinforcement learning "from scratch" can rediscover, and sometimes refine, the nuanced strategies of Blackjack. Double Q-Learning [6], in particular, offers a blueprint for handling challenging, high-variance, partially observable decision tasks. As RL systems become more powerful, such explorations will continue to bridge the gap between algorithms and the skills that have long defined successful play—at the tables and beyond.

REFERENCES

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2005, vol. 1.

[2] P. A. Griffin, *The Theory of Blackjack: The Complete Card Counter's Guide to the Casino Game of 21*, 6th ed. Huntington Press, 1999.

[3] S. N. Ethier, *The Doctrine of Chances: Probabilistic Aspects of Gambling*. Springer-Verlag, 2010.

[4] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[5] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[6] H. van Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23 (NeurIPS)*, 2010, pp. 2613–2621.

[7] A. Snyder, *The Big Book of Blackjack*. Cardoza Publishing, 2006.

[8] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the Fourth Connectionist Models Summer School*, 1993, pp. 255–263.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.